НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

Теплоенергетичний факультет

Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено
Завідувач кафедри
_____ Олександр Коваль
(підпис) (ініціали, прізвище)
“___”_____2020р.

ДИПЛОМНА РОБОТА

на здобуття ступеня бакалавра

з напряму підготовки 122 Комп’ютерні науки та інформаційні технології

на тему Інтерактивна карта сонячного гарячого водопостачання по областях України

Виконав (-ла): студент (-ка) 4 курсу, групи ТМ_62

Дукач Станіслав Васильович _____
(прізвище, ім’я, по батькові) (підпис)

Керівник доцент кафедри, к.в.н. доцент Андрій Онисько _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Консультант _____ _____ _____
(назва розділу) (вчені ступінь та звання, прізвище, ініціали) (підпис)

Рецензент_____ _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі
немає запозичень з праць інших авторів
без відповідних посилань.
Студент _____
(підпис)

Київ – 2020 року

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший рівень

Напрям підготовки 122 Комп’ютерні науки та інформаційні технології

Спеціалізація Інформаційні технології моніторингу довкілля

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Олександр Коваль
(підпис)
”___”_____2020р.

ЗАВДАННЯ
на дипломну роботу студенту
Дукача Станіслава Васильовича
(прізвище, ім’я, по батькові)

1. Тема роботи Інтерактивна карта сонячного гарячого водопостачання по областях України

керівник роботи доцент кафедри, к.в.н. доцент Андрій Онисько
(прізвище, ім’я, по батькові науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від ”__”___ 202__р. № __

2. Строк подання студентом роботи _____
3. Вихідні дані до роботи React.js проект

4.Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити проаналізувати існуючі програмні рішення та засоби аналізу сонячної активності, спроектувати архітектуру системи аналізу сонячної активності, розробити програмне забезпечення, розробити інтерфейс користувача

5. Перелік ілюстративного матеріалу
1. Актуальність 2. Мета та завдання роботи 3. Функції додатку 4. Опис функціональності системи 5. Архітектура програмного комплексу 6. Опис структури інтерфейсу 7. Використані програмні засоби 8. Інтерфейс головної сторінки 8. Калькулятор сонячної енергії 9. Висновки

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1	Матях С.В.		

7. Дата видачі завдання ”___”_____201___ р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Затвердження теми роботи		
2.	Вивчення та аналіз задачі	25.02.2020	
3.	Розробка архітектури та загальної структури системи	10.04.2020	
4.	Розробка структур окремих підсистем	15.04.2020	
5.	Програмна реалізація системи	25.04.2020	
6.	Оформлення пояснювальної записки	25.05.2020	
7.	Захист програмного продукту		
8.	Передзахист		
9.	Захист		

Студент _____ _____
(підпис) (прізвище та ініціали,)

Керівник роботи _____ _____
(підпис) (прізвище та ініціали,)

АНОТАЦІЯ

Метою роботи було створення модуля клієнта, який дозволяє розрахувати рентабельність установки сонячних батарей та сонячних панелей. Система дозволяє юзеру вибрати на карті України будь-які точки, для яких проводиться розрахунок виробництва енергії з використанням сонячного випромінювання. Додаток дозволяє вибирати різні типи сонячних панелей з різними параметрами та вводити додаткові параметри. Додаток - це інтерактивна карта України з даними про сонячну інсоляцію за регіонами, демонстрація розрахунків виробництва електроенергії для сонячних батарей та гарячої води для сонячних систем та відображення результатів у вигляді графіка.

Записка містить 67 сторінок, 10 картинок, 6 таблиць

ABSTRACT

The aim of the work was to create a customer module that allows you to calculate the profitability of the installation of solar panels and solar panels. The system allows the user to select on the map of Ukraine any points for which energy production is calculated using solar radiation. The application allows you to select different types of solar panels with different parameters and enter additional parameters. The appendix is an interactive map of Ukraine with data on solar insolation by region, demonstration of calculations of electricity production for solar panels and hot water for solar systems and display of results in the form of a graph.

The note contains 67 pages, 10 pictures, 6 tables

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API (англ. Application Programming Interface) — прикладний програмний інтерфейс;

БД — база даних;

CRUD (англ. create read update delete) — 4 базові функції управління даними «створення, зчитування, зміна і видалення»;

SQL (англ. Select query language) — декларативна мова програмування для взаємодії користувача з базами даних;

Фреймворк — заготовки, шаблони для програмної платформи, що визначають архітектуру програмної системи; програмне забезпечення, що полегшує розробку і об'єднання різних модулів програмного проекту.

ВСТУП

Ідея отримання енергії з невичерпного джерела завжди була актуальною. З часом стали з’являтися все новіші альтернативні джерела енергії. Вони почали розвиватися і приносити все більше і більше користі. Одним з таких джерел є сонце.

Сонячна енергетика є перспективним напрямком відновлювальної енергетики в Україні. Важливою перевагою використання енергії сонячної радіації, крім невичерпності її потоку та екологічності, є можливість її використання у віддалених районах земної поверхні з можливістю її прямого перетворення в теплову та електричну енергію.

Раніше обладнання було занадто дорогим та його ефективність була досить низькою. Тому альтернативна енергетика могла бути використана лише шляхом великих коштів. Але сучасне обладнання набагато ефективніше. Це дозволяє отримувати сонячну енергію не тільки великими масштабами, але й для домашнього користування. Але перш ніж зважитися на збільшення енергії сонця, потрібно проаналізувати багато факторів, які впливають на переваги, які отримує споживач.

Тому було запропоновано створити веб-додаток, який надасть людям можливість отримати приблизні дані про те, наскільки вигідним буде встановлення певного типу сонячних батарей або сонячних систем.

Для вивчення та розрахунку рентабельності сонячних електростанцій та сонячних систем було запропоновано розробити програмну систему, яка демонструватиме сонячну активність у регіонах та зможе розраховувати щомісячне виробництво електроенергії за допомогою сонячних батарей та гарячої води із використанням сонячних систем .

Це рішення допоможе людям, які зацікавлені отримати енергію від сонця для своїх побутових цілей, але не знають, який тип системи їм потрібен і як це буде корисним для їх регіону.

Розроблена програма в моделюванні сигналу враховує такі особливості, як

інтенсивність сонячної інсоляції залежно від обраного регіону та сезону, вибирається кут, під яким падають сонячні промені залежно від пори року. Дозволяє вибирати різні типи систем для перетворення сонячної енергії в електричну і теплову. Цей модуль надає можливість виконувати обчислення для отримання інформації у вигляді таблиці.

Записка містить 5 розділів.

У першому розділі описується постановка задачі відображення сонячної активності на території України.

У другому розділі описується характеристика сонячної радіації, сонячні батареї, технічно досяжний потенціал сонячної енергії.

У третьому розділі описані засоби реалізації програмного продукту.

У четвертому розділі описана робота користувача з системою.

У п’ятому розділі висновки.

Опис проблеми та постановка задачі

Використання сучасних сонячних колекторів забезпечує високий рівень поглинання сонячної енергії та стабільність гарячого водопостачання протягом року по всій Україні. На сучасному етапі розвитку сонячної теплової енергії на першому місці стоять проблеми ефективного використання енергії сонячної радіації за рахунок використання передових технологій та встановлення оптимальних параметрів енергетичного обладнання.

Процедура визначення ефективності сонячних систем гарячого водопостачання, представлена в роботі, передбачає енергетичні та економічні параметри сонячного теплового обладнання в певній місцевості, визначаючи тип та параметри сонячних систем для їх найбільш ефективного використання.

Вибір типу та продуктивності сонячних колекторів для певної місцевості в першу чергу орієнтований на потреби конкретного споживача та конкретні показники сонячної радіації в даній області (середньомісячна та середньорічна кількість прямого, розсіяного та загального сонячного випромінювання). На підставі представлених даних визначається знижена добова інтенсивність поглинання сонячної радіації сонячним колектором з урахуванням робочих параметрів сонячної установки та оптимального кута нахилу до горизонту.

Розраховані параметри енергії додатково використовуються для встановлення економічної ефективності, періоду окупності сонячної установки та екологічної ефективності за рахунок зменшення викидів вуглекислого газу. Сонячне теплопостачання в Україні має достатній досвід використання та розроблену нормативну базу для проектування, а технологічний потенціал галузі дозволяє вирішити проблему масового виробництва геліотехнічного обладнання.

Запропонований порядок оперативного встановлення ефективності впровадження систем сонячного гарячого водопостачання для потенційних споживачів сприятиме широкому розвитку сонячної теплової енергії по всій Україні та, відповідно, скоротить використання викопного палива та покращить навколишнє середовище

Технічно-досяжний потенціал сонячної енергії

Технічно-досяжний потенціал сонячної енергії регіону – це середня багаторічна сумарна енергія, що може бути отримана в регіоні від сонячного випромінювання протягом одного року при сучасному рівні розвитку науки і техніки та при дотриманні екологічних норм.

Технічно-досяжний потенціал сонячної енергії являє собою суму технічно-досяжних потенціалів теплової та електричної енергії, що одержуються відповідним перетворенням сонячного випромінювання.

Технічно-досяжний потенціал регіону являє собою суму технічно-досяжних потенціалів складових його зон. Для кожної зони використовуються наступні позначення:

$W_{ТД}$, кВт·год/рік – технічно-досяжний потенціал сонячної енергії;

$W_{ТДТ}$, кВт·год/рік – технічно-досяжний потенціал теплової енергії від сонячного випромінювання;

$W_{ТДЕ}$, кВт·год/рік – технічно-досяжний потенціал електроенергії від сонячного випромінювання [1]:

$$W_{ТД} = W_{ТДТ} + W_{ТДЕ}. \qquad (2.1)$$

Площа, яка з господарських та екологічних міркувань є доцільною для використання сонячної енергії (S_C, м^2) дорівнює частині q загальної площі S, що залишається після вирахування площ лісів, парків, сільськогосподарських угідь та інших територій, на яких розміщення установок ускладнене або заборонене:

$$S_C = qS. \qquad (2.2)$$

Від загальної площі, доцільної для використання сонячної енергії, визначається частка площі S_C, доцільна для установки сонячних теплових колекторів k_T, та частка площі S_C, доцільна для установки сонячних фотоелектричних батарей k_E:

$$k_T + k_E = 1. \qquad (2.3)$$

Значення q, k_T, k_E є специфічними для кожної зони; у той же час, на основі досвіду деяких промислово розвинених країн $q \le 0{,}01$ [38].

Технічно-досяжний потенціал теплової енергії від сонячного випромінювання визначається за формулою [1]:

$$W_{ТДТ} = \sum W_{ТДТi},\ i = 1, 2, \ldots, 12, \qquad (2.4)$$

де підсумовування відбувається по всіх місяцях року; технічний потенціал i-го місяця:

$$W_{ТДТ\,i} = E_i \cdot k_T \cdot q \cdot S \cdot F \cdot \left[(\tau\alpha) - U_{L.} \cdot (T - T_{Oi}) \cdot \cos(\varphi - \delta) \cdot \frac{t_{Ci}}{E_i} \right], \qquad (2.5)$$

де Ei – прихід сонячної радіації на одиницю горизонтальної поверхні в і-му місяці;

k_T – доля площі доцільна для установки теплового колектора;

q – температура гарячої води

S – площа, доцільна для використання сонячної енергії;

F – значення параметра сонячного колектора, що характеризує його технічний рівень;

T_o –середньорічна температура вдень, під час роботи сонячної установки;

T_{oi} – середньомісячна температура протягом місяця під час роботи сонячної установки;

$(\varphi - \delta)$ – кут нахилу колектора до Землі (максимальна необхідна площа колекторів дорівнює $k_T \cdot q \cdot S \cdot \cos(\varphi - \delta)$; t_{Ci}; год/міс. – термін роботи колекторів (число сонячних годин у місяці).

При проведенні розрахунку фіксуються вихідні дані: k_T, q, температура гарячої води, значення параметрів сонячних теплових колекторів, що характеризують сучасний технічний рівень: $F(\alpha\tau) = 0{,}9$; $FU_L = 0{,}005$ кВт/(м·°С); експериментально визначені середньомісячні температури T_{Oi} для $i = 1{,}2, \ldots, 12$; кут δ і термін роботи колекторів t_{Ci}.

В таблиці представлено показники технічно-досяжного енергетичного потенціалу сонячної енергії в кожній із областей України, у тому числі технічно-досяжні показники енергетичного потенціалу сонячної енергії для виробництва теплової та електричної енергії.

В результаті обробки статистичних метеорологічних даних щодо надходження сонячної радіації визначено питомі енергетичні показники сонячної енергії та розподіл енергетичного потенціалу сонячного випромінювання для кожної з областей України. Річний технічно-досяжний енергетичний потенціал сонячної енергії в Україні є еквівалентним 4,2 млн т н.е., а його використання дозволяє заощадити біля 5 млрд м^3 природного газу.

Наведені енергетичні показники щодо надходження сонячної радіації є базовими для вибору сонячного енергетичного обладнання та для встановлення його оптимальної потужності, а розподіл середньорічної сумарної сонячної радіації дає загальне уявлення щодо доцільності розміщення сонячного енергетичного обладнання в конкретній місцевості.

Середньорічна кількість сумарної сонячної радіації, що надходить на 1 м^2 поверхні на території України, знаходиться в межах від 1070 кВт·год/м^2 у північній частині України до 1400 кВт·год/м^2 в АР Крим.

Потенціал сонячної енергії в Україні є достатньо високим для широкого впровадження як фотоелектричного, так і теплоенергетичного обладнання практично в усіх областях.

Фотоелектричне обладнання може достатньо ефективно експлуатуватися протягом усього року, а термін ефективної експлуатації геліоенергетичного обладнання у південних областях України – 7 місяців (з квітня до жовтня), у північних областях – 5 місяців (з травня до вересня).

У кліматометеорологічних умовах України для сонячного теплопостачання ефективним є застосування пласких сонячних колекторів, які

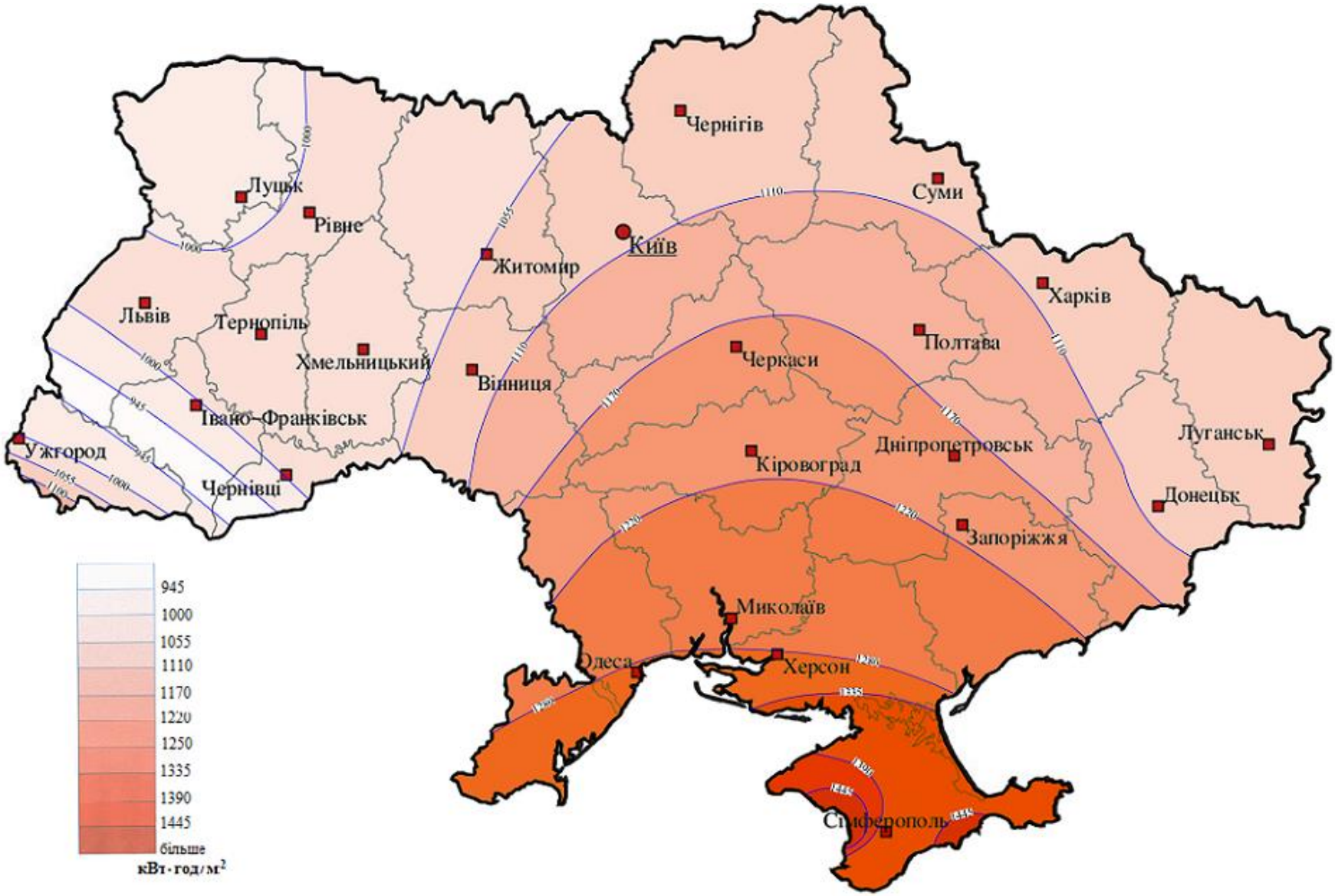
використовують як пряму, так і розсіяну сонячну радіацію.

Для регіонів із малосніжними зимами та великою кількістю сонячних днів протягом року (АР Крим, Причорноморський та Приазовський регіони тощо) ефективним є застосування сонячних колекторів з вакуумними трубками.

Перетворення сонячної енергії в електричну в умовах України доцільно орієнтувати в першу чергу на використання фотоелектричних пристроїв.

Таблиця Технічно-досяжний енергетичний потенціал сонячної енергії на території України

№ з/п	Області	Технічно-досяжний енергетичний потенціал				№ з/п	Області	Технічно-досяжний енергетичний потенціал			
		електричний		тепловий ($\times 10^5$), т н.е./рік	загальний ($\times 10^5$), т н.е./рік			електричний		тепловий ($\times 10^5$), т н.е./рік	загальний ($\times 10^5$), т н.е./рік
		млрд кВт· год/рік	т н.е./рік ($\times 10^5$)					млрд. кВт· год/рік	т н.е./ рік ($\times 10^5$)		
1.	АР Крим	2,20	1,9	0,8	2,7	14.	Миколаївська	1,51	1,3	0,6	1,9
2.	Вінницька	1,40	1,2	0,5	1,7	15.	Одеська	2,09	1,8	0,8	2,6
3.	Волинська	1,04	0,9	0,4	1,3	16.	Полтавська	1,51	1,3	0,5	1,8
4.	Дніпропетровська	1,86	1,6	0,6	2,2	17.	Рівненська	0,93	0,8	0,4	1,2
5.	Донецька	1,51	1,3	0,6	1,9	18.	Сумська	1,28	1,1	0,4	1,5
6.	Житомирська	1,51	1,3	0,5	1,8	19.	Тернопільська	0,81	0,7	0,3	1,0
7.	Закарпатська	0,81	0,7	0,3	1,0	20.	Харківська	1,62	1,4	0,6	2,0
8.	Запорізька	1,62	1,4	0,6	2,0	21.	Херсонська	1,74	1,5	0,7	2,2
9.	Івано-Франківська	0,70	0,6	0,3	0,9	22.	Хмельницька	1,16	1,0	0,4	1,4
10.	Київська	1,51	1,3	0,5	1,8	23.	Черкаська	1,28	1,1	0,4	1,5
11.	Кіровоградська	1,28	1,1	0,5	1,6	24.	Чернівецька	0,46	0,4	0,2	0,6
12.	Луганська	1,51	1,3	0,6	1,9	25.	Чернігівська	1,62	1,4	0,6	2,0
13.	Львівська	1,28	1,1	0,4	1,5	**ВСЬОГО**		**34,24**	**29,5**	**12,5**	**42,0**


Чернігів
Суми
Луцьк
Рівне
Київ
Житомир
Харків
Львів
Тернопіль
Полтава
Хмельницький
Черкаси
Вінниця
Івано-Франківськ
Луганськ
Ужгород
Дніпропетровськ
Кіровоград
Чернівці
Донецьк
Запоріжжя
Миколаїв
Одеса
Херсон
Сімферополь
945
1000
1055
1110
1170
1220
1250
1335
1390
1445
більше
кВт·год/м²

Розподіл питомої сумарної сонячної радіації на території України протягом року

(Національний атлас України. – К.: ДНВП «Картографія», 2007)

Нафтовий еквівалент (англ. *oil equivalent*) – стандартизована одиниця вимірювання енергії, що використовується для порівняння використання великої кількості енергії з різних джерел.

1 т н.е. (toe) еквівалентна кількості енергії, що виділяється при спалюванні однієї тонни сирої нафти, близько 41,868 ГДж або 11,63 МВт.год. енергії.

Для переведення однієї тонни нафтового еквіваленту в одну тонну вугільного еквіваленту (т у.п.) застосовують коефіцієнт 1,428.

За допомогою використання показника "паливо умовне" або «нафтовий еквівалент» складаються паливні баланси або загальні енергетичні баланси галузей, країн та світу в цілому.

В кліматометеорологічних умовах України використання теплової енергії сонячної радіації є найбільш ефективним для нагріву води в сонячних колекторах з подальшим її використанням на різні потреби.

В таблиці представлено розподіл технічно-досяжного потенціалу сонячного гарячого водопостачання по областях України.

1 т н.е. (toe) = 11,63 МВт.год.

Таблиця Потенціал сонячного гарячого водопостачання по областях України

№ з/п	Області	Технічно-досяжний потенціал сонячної енергії для виробництва теплової енергії, т н.е./рік (× 10^5)	Потенціал сонячного гарячого водопостачання, МВт.год
1.	АР Крим	0,8	9,3
2.	Вінницька	0,5	
3.	Волинська	0,4	
4.	Дніпропетровська	0,6	
5.	Донецька	0,6	
6.	Житомирська	0,5	
7.	Закарпатська	0,3	
8.	Запорізька	0,6	

9.	Івано-Франківська	0,3	
10.	Київська	0,5	
11.	Кіровоградська	0,5	
12.	Луганська	0,6	
13.	Львівська	0,4	
14.	Миколаївська	0,6	
15.	Одеська	0,8	
16.	Полтавська	0,5	
17.	Рівненська	0,4	
18.	Сумська	0,4	
19.	Тернопільська	0,3	
20.	Харківська	0,6	
21.	Херсонська	0,7	
22.	Хмельницька	0,4	
23.	Черкаська	0,4	
24.	Чернівецька	0,2	
25.	Чернігівська	0,6	
ВСЬОГО		**12,5**	

В результаті обробки статистичних метеорологічних даних щодо надходження сонячної радіації визначено конкретні енергетичні показники сонячної енергії та розподіл енергетичного потенціалу сонячної радіації для кожного з регіонів

України. Щорічно технічно досяжний енергетичний потенціал сонячної енергії в Україні еквівалентний 4,2 мільйона тонн н.е., а його використання заощаджує близько 5 млрд. М3 природного газу.

Ці показники енергетичності для сонячної радіації є основою для підбору обладнання для сонячної енергії та для встановлення його оптимальної потужності, а розподіл середньорічної загальної сонячної радіації дає загальне уявлення про можливість розміщення обладнання сонячної енергії у певній місцевості. Середньорічна кількість загальної сонячної радіації на 1 м2 поверхні в Україні знаходиться в межах від 1070 кВт / год на м2 у північній частині України до 1400 кВт / год в м2 в Автономній Республіці Крим.

Потенціал сонячної енергії в Україні досить високий для широкого впровадження як фотоелектричного, так і теплоенергетичного обладнання майже у всіх районах. Фотоелектричне обладнання можна експлуатувати досить ефективно протягом року, а термін ефективної роботи сонячної енергетичної апаратури у південних регіонах України - 7 місяців (з квітня по жовтень), у північних регіонах - 5 місяців (з травня по вересень) .

В кліматичних та метеорологічних умовах України ефективним є використання плоских сонячних колекторів, які використовують як пряму, так і розсіяну сонячну радіацію.

Для регіонів з безсніжними зимами та великою кількістю сонячних днів протягом року (Крим, Причорномор’я та Приазов'я тощо) ефективно використовувати сонячні колектори з вакуумними трубами.

Перетворення сонячної енергії в електроенергію в умовах України має бути зосереджено насамперед на використанні фотоелектричних пристроїв.

Сонячні батареї

Сонячні панелі використовують сонячне світло як джерело енергії для отримання електроенергії. Фотоелектричний (PV) модуль - це набір елементів, з'єднаних один з одним, як правило, що складається з 6-10 фотоелектричних сонячних батарей. Фотоелектричні модулі складають фотоелектричну систему фотоелектричної системи, яка виробляє та постачає сонячну електрику в комерційних та житлових приміщеннях.
Фотоелектричні модулі використовують світлову енергію від сонця для отримання електрики завдяки фотоефекту. Більшість модулів використовують кристалічні клітини на основі кремнію або клітини тонкої плівки. Структурним елементом модуля може бути верхній або задній шар. Клітини також повинні бути захищені від механічних пошкоджень і вологи. Більшість модулів є жорсткими, але також є напів гнучкі на основі тонкоплівкових елементів. Елементи повинні бути з'єднані електрично послідовно, один до одного.
Деякі спеціальні сонячні фотоелектричні модулі містять концентратори, у яких світло фокусується лінзами або дзеркалами на менших осередках. Це дозволяє використовувати клітини з високою витратою на одиницю площі (наприклад, арсенід галію) економічно ефективним способом.
Кожен модуль має номінальну вихідну потужність постійного струму, яка зазвичай становить від 100 до 365 Вт (Вт). ККД модуля визначається площею модуля при однаковій номінальній потужності - модуль 230 Вт з ККД 8% матиме вдвічі більше площі модуля 230 Вт з ККД 16%. Існує кілька комерційно доступних сонячних модулів з ефективністю понад 24%.
Залежно від конструкції, фотоелектричні модулі можуть генерувати електроенергію з частотного діапазону світла, але зазвичай не можуть охопити весь сонячний спектр (зокрема, ультрафіолетове, інфрачервоне та тьмяне чи дифузне світло). Тому більша частина енергії падаючого сонячного світла втрачається сонячними модулями. Один сонячний модуль може виробляти лише обмежену кількість енергії; більшість установок містять кілька модулів. Фотоелектрична система, як правило, включає

масив фотоелектричних модулів, інвертор, акумуляторну батарею, з'єднувальну проводку і, необов'язково, механізм відстеження сонячної енергії.

Висновки до розділу

У даному розділі було розглянуто актуальність сонячної енергетики. Було зроблено аналіз того, які типи систем використовуються для перетворення енергії та які їх характеристики. Було досліджено алгоритми підрахунку електроенергії та гарячої води, отриманих від сонячних панелей та геліостанцій відповідно, та обрано потрібні формули, які будуть використовуватись у веб-карті.

ЗАСОБИ РЕАЛІЗАЦІЇ ПРОГРАМНОЇ СИСТЕМИ

Проаналізувавши сучасні методи створення програмного забезпечення та задачу створення інтерактивної карти України стало зрозуміло, що клієнт-серверна архітектура є найбільш доцільною так як вона передбачає можливість єдиної системи управління та бази даних для всіх користувачів, а це є однією із головних задач розробки «софту», всі мають бути в рівних умовах задля запобігання маніпуляції даними та коректної роботи сервісу.

Для того, щоб охопити найбільшу кількість потенційних користувачів які будуть мати необхідність використовувати розроблену мною систему, постало питання вибору платформи для запуску системи. Було розглянуто всі популярні операційні системи, як комп’ютерні так і мобільні, та проведено статистичний аналіз кількості користувачів які підпадають під критерії потенційного користувача даним продуктом. Отримані дані були не задовільними, кожна з систем мала якісь обмеження та не могла відповідати всім вимогам які було поставлено. Стало зрозуміло, щоб надати зручний доступ до системи та однаковий для всіх користувачів, потрібно знайти область в якій всі ці системи пересікаються. Веб-браузер є спосіб надати можливість користування сервісом, всім в кого є доступ до сучасних операційних систем.

Вибір архітектури програмного комплексу

Для вирішення поставленої задачі нам необхідно розробити наступні модулі:

- Клієнт – являється інтерфейсом для взаємодії користувача з іншими модулями, а конкретно з сервером, використовуючи його API(application programming interface).
- Сервер – середня ланка в архітектурі, основною задачею серверу, є надання користувачу доступу до даних в базі даних та керування ними.
- База даних – забезпечує інтерфейс для роботи з даними, гарантує їх цілісність та спрощує розробку архітектури та способу збереження на носію інформації.

Вище приведені модулі приводять нас до триланкової архітектури побудови продукту, яка вигляє наступним чином, як на Рис. 5.1

Ми створюємо централізовану систему, це означає, що найважливішою ланкою нашої системи є сервер. Він надає інтерфейс для користування клієнту, тобто, він має набір певних команд які потрібно виконати щоб задовольнити вимоги поставлені до функціоналу продукту.

Також сервер контролює та перевіряє всі запити від клієнту, в цілях безпеки ми не можемо допустити, щоб до всіх можливостей серверу мав доступ будь-хто, тому на сервері створена перевірка на аутентифікацію яку повинен проходи кожен запит. Лише два запити можливо провести без цієї перевірки:

1. Авторизація
2. Реєстрація

Після того як запит пройшов перевірку ми можемо його пропустити далі, це означає, що сервер може виконувати операції з базою даних.

Сам користувач має безпосередній доступ лише до клієнтської частини продукту, створений веб-сервіс надає елементи керування за допомогою яких

здійснюється взаємодія з картою, перегляду даних в зручному та візульально приємному і зрозумілому форматі. На клієнті також передбачена система аутентифікації, тільки після того як користувач зробив запит зі своїми попердньо зареєстрованими даними в системі і отримав унікальний ключ доступу, отримує можливість користування функціоналом.

Головною задачею рівня бази даних є забезпечення збереження даних, які сервер зберігає для подальшого використання. Крім того, забезпечується цілісність даних за допомогою зовнішніх звя’зків та ключів. На рівні бази даних також можна реалізовувати деяку бізнес-логіку, яка не потребує використання зовнішніх джерел даних окрім самої бази даних та її таблиць.

Опис архітектури серверу

ООП (об'єктно-орієнтоване програмування) стало невід'ємною частиною розробки багатьох сучасних проектів, але, не дивлячись на популярність, ця парадигма є далеко не єдиною.

Перш за все варто відповісти, навіщо? Об'єктно-орієнтована ідеологія розроблялася як спроба пов'язати поведінку суті з її даними і спроектувати об'єкти реального світу і бізнес-процесів в програмний код. Замислювалося, що такий код простіше читати і розуміти людиною, так як Людям властиво сприймати навколишній світ як безліч взаємодіючих між собою об'єктів, що піддаються певної класифікації. Чи вдалося ідеологам досягти мети, однозначно відповісти складно, але де-факто ми маємо масу проектів, в яких з програміста вимагатимуть ООП.

Не слід думати, що ООП якимось дивним чином прискорить написання програм. У більшості випадків це не так, і час економиться не на стадії розробки, а на етапах підтримки (розширення, модифікація, налагодження і тестування), тобто довгостроковій перспективі. Якщо вам потрібно написати одноразовий скрипт, який не потребує подальшої підтримки, то і ООП в цьому завданні, найімовірніше, не стане в нагоді. Однак, значну частину життєвого циклу більшості сучасних проектів складають саме підтримка і розширення. Саме по собі наявність ООП не робить вашу архітектуру бездоганною, і може навпаки призвести до зайвих ускладнень.

Іноді можна зіткнутися з критикою на адресу швидкодії ООП-програм. Це правда, незначний «оверхед» присутній, але настільки незначний, що в більшості випадків їм можна знехтувати на користь переваг. Проте, у вузьких місцях, де в одному потоці повинні створюватися або оброблятися мільйони об'єктів в секунду, варто як мінімум переглянути необхідність ООП, бо навіть мінімальний «оверхед» в таких кількостях може відчутно вплинути на продуктивність. Профілювання допоможе вам зафіксувати різницю і прийняти рішення. В інших же випадках, скажімо, де левова частка швидкодії впирається в IO, відмова від об'єктів буде передчасною оптимізацією.

Класи і об'єкти

Відразу ліричний відступ: об'єктно-орієнтований підхід можливий і без класів, але ми будемо розглядати, класичну схему.

Найпростіше пояснення: клас - це креслення об’єкту, а екземпляри цього класу - конкретні різновиди цих об’єктів. І хоча вони і зібрані по одному кресленням, вміють однаково функціонувати, вони володіють власним унікальним станом. Стан - це ряд мінливих властивостей. Тому у двох різних об'єктів одного класу ми можемо спостерігати різний ім'я, вік, місце розташування, рівень заряду і так далі. Саме наявність цих властивостей і їх типи описуються в класі.

Таким чином, клас - це опис того, якими властивостями і поведінкою буде володіти об'єкт. А об'єкт - це екземпляр з власним станом цих властивостей.

Ми говоримо «властивості і поведінку», але звучить це якось абстрактно і незрозуміло. Звичніше для програміста буде звучати так: «змінні і функції». Насправді «властивості» - це такі ж звичайні змінні, просто вони є атрибутами якогось об'єкта (їх називають полями об'єкта). Аналогічно «поведінку» - це функції об'єкта (їх називають методами), які теж є атрибутами об'єкта. Різниця між методом об'єкта і звичайною функцією лише в тому, що метод має доступ до свого стану через поля.

Розглянемо основні елементи класу:

this - це спеціальна локальна змінна (всередині методів), яка дозволяє об'єкту звертатися зі своїх методів до власних атрибутів. Звертаю увагу, що тільки до

власних. Якщо зовні звернення буде виглядати так: obj.x, то зсередини, якщо об’єкт захоче сам звернутися до свого поля x, в його методі звернення буде звучати так: this.x. У більшості мов ця змінна називається this, але трапляються й винятки (наприклад, self)

constructor - це спеціальний метод, який автоматично викликається при створенні об'єкта. Конструктор може приймати будь-які аргументи, як і будь-який інший метод. У кожній мові конструктор позначається своїм ім'ям. Десь це спеціально зарезервовані імена типу __construct або __init__, а десь ім'я конструктора має збігатися з ім'ям класу. Призначення конструкторів - провести первинну ініціалізацію об'єкта, заповнити потрібні поля.

new - це ключове слово, яке необхідно використовувати для створення нового екземпляра будь-якого класу. У цей момент створюється об'єкт і викликається конструктор. Ключове слово new в деяких мовах відсутня, і конструктор викликається автоматично при спробі викликати клас як функцію, наприклад так: Object().

Метод constructor працюють з внутрішнім станом, а в усьому іншому не відрізняються від звичайних функцій. Навіть синтаксис оголошення збігається.

Класи можуть мати методами, яким не потрібно стан і, як наслідок, створення об'єкта. У цьому випадку метод роблять статичним.

SRP

(Single Responsibility Principle / Принцип єдиної відповідальності / Перший принцип SOLID). З ним ви, напевно, вже знайомі з інших парадигм: «одна функція повинна виконувати тільки одне закінчене дію». Цей принцип справедливий і для класів: «Один клас повинен відповідати за якусь одну задачу». На жаль з класами складніше визначити грань, яку потрібно перетнути, щоб принцип порушувався.

Існують спроби формалізувати цей принцип за допомогою опису призначення класу одним реченням без союзів, але це дуже спірна методика, тому довіртеся своїй інтуїції і не кидайтеся в крайнощі. Не потрібно робити з класу швейцарський ніж, але і плодити мільйон класів з одним методом всередині - теж нерозумно.

Асоціація

Традиційно в полях об'єкта можуть зберігатися не тільки звичайні змінні стандартних типів, а й інші об'єкти. А ці об'єкти можуть в свою чергу зберігати якісь інші об'єкти і так далі, утворюючи дерево (іноді граф) об'єктів. Це відношення називається асоціацією.

1. Композиція - випадок, коли життєвий цикл дочірнього об'єкта збігається з життєвим циклом батьківського.

2. Агрегація - випадок, коли життєвий цикл дочірнього об'єкта не залежить від життєвого циклу батьківського, і може використовуватися іншими об'єктами.

ООП будується на фундаментальній трійці - інкапсуляція, поліморфізм і успадкування, на яких грунтується весь об'єктно-орієнтований підхід. Розберемо їх по порядку.

спадкування

Спадкування - це механізм системи, який дозволяє, як би парадоксально це не звучало, успадковувати одними класами властивості і поведінку інших класів для подальшого розширення або модифікації.

Що якщо, ми не хочемо штампувати однакові об’єкти, а хочемо зробити загальний каркас, але з різним обвісом? ООП дозволяє нам таку витівку шляхом поділу логіки на подібності та відмінності з подальшим винесенням подібностей в батьківський клас, а відмінностей в класи-нащадки.

перевантаження

Якщо ж в класі-нащадку перевизначити вже існуючий метод в класі-батьку, то спрацює перевантаження. Це дозволяє не доповнювати поведінку батьківського класу, а модифікувати. У момент виклику методу або звернення до полю об'єкта, пошук атрибута походить від нащадка до самого кореня - батькові.

недоречне застосування

Цікаво, що надмірно глибока ієрархія наслідування може призвести до зворотного ефекту - ускладнення при спробі розібратися, хто від кого успадковується, та який метод в якому разі викликається. До того ж, не всі архітектурні вимоги можна реалізувати за допомогою наслідування. Тому застосовувати спадкування слід без фанатизму. Існують рекомендації, що

закликають віддавати перевагу композицію спадкоємства там, де це доречно. Будь-яка критика успадкування, яку я зустрічав, підкріплюється невдалими прикладами, коли спадкування використовується в якості золотого молотка. Але це зовсім не означає, що спадкування в принципі завжди шкодить.

Як при описі відносин двох сутностей визначити, коли доречно спадкування, а коли - композиція? Можна скористатися популярною шпаргалкою: запитайте себе, сутність А є сутністю Б? Якщо так, то швидше за все, тут підійде успадкування. Якщо ж сутність А є частиною сутності Б, то наш вибір - композиція.

спадкування статично

Ще одна важлива відмінність спадкування від композиції в тому, що успадкування має статичну природу і встановлює відносини класів тільки на етапі інтерпретації / компіляції. Композиція ж, як ми бачили в прикладах, дозволяє змінювати ставлення сутностей на льоту прямо в Рантайм - іноді це дуже важливо, тому про це треба пам'ятати при виборі відносин (якщо звичайно немає бажання використовувати метапрограмування).

множинне спадкування

Ми розглянули ситуацію, коли два класи успадковані від загального нащадка. Але в деяких мовах можна зробити і навпаки - успадкувати один клас від двох і більше батьків, об'єднавши їх властивості та поведінку. Можливість успадковуватися від декількох класів замість одного - це множинне спадкування.

Взагалі, в колах ілюмінатів існує думка, що множинне спадкування - це гріх, воно несе за собою ромбовидну проблему і плутанину з конструкторами. Крім того, завдання, які вирішуються множинним спадкуванням, можна вирішувати іншими механізмами, наприклад, механізмом інтерфейсів (про який ми теж поговоримо). Але справедливості заради, слід зазначити, що множинне спадкування зручно використовувати для реалізації домішок.

абстрактні класи

Крім звичайних класів в деяких мовах існують абстрактні класи. Від звичайних класів вони відрізняються тим, що не можна створити об'єкт такого класу. Навіщо ж потрібен такий клас, запитає читач? Він потрібен для того, щоб від

нього могли успадковуватися нащадки - звичайні класи, об'єкти яких уже можна створювати.

Абстрактний клас поряд зі звичайними методами містить в собі абстрактні методи без імплементації (з сигнатурою, але без коду), які зобов'язаний імплементувати програміст, який задумав створити клас-нащадок. Абстрактні класи не є обов'язковими, але вони допомагають встановити контракт, який зобов'язує імплементувати певний набір методів, щоб уберегти програміста з поганою пам'яттю від помилки імплементації.

поліморфізм

Поліморфізм - властивість системи, що дозволяє мати безліч реалізацій одного інтерфейсу.

інкапсуляція

Інкапсуляція - це контроль доступу до полів і методів об'єкта. Під контролем доступу мається на увазі не тільки можна / неможна, але і різні валідації, підгрузки, обчислення та інше динамічну поведінку.

У багатьох мовах частиною інкапсуляції є приховування даних. Для цього існують модифікатори доступу (опишемо ті, які є майже у всіх ООП мовами):

public - до атрибуту може отримати доступ будь-який бажаючий

private - до атрибуту можуть звертатися тільки методи даного класу

protected - те саме, що і private, тільки доступ отримують і спадкоємці класу в тому числі

Як правильно вибрати модифікатор доступу? У найпростішому випадку так: якщо метод повинен бути доступний зовнішньому коду, вибираємо public. В іншому випадку - private. Якщо є успадкування, то може знадобитися protected в разі, коли метод не повинен викликатися зовні, але повинен викликатися нащадками.

Аксессор (геттери і сеттери)

Геттери і сеттери - це методи, завдання яких контролювати доступ до полів. Геттер зчитує і повертають значення поля, а сетер - навпаки, приймає в якості аргументу значення і записує в поле. Це дає можливість забезпечити такі методи

додатковими обробками. Наприклад, сетер під час запису значення в поле об'єкта, може перевірити тип, або входить значення в діапазон допустимих (валідація). У геттер ж можна додати, ледачу ініціалізацію або кешування, якщо актуального значення насправді лежить в базі даних. Застосувань можна придумати безліч.

У деяких мовах є синтаксичний цукор, що дозволяє такі аксессор маскувати під властивості, що робить доступ прозорим для зовнішнього коду, який і не підозрює, що працює не з полем, а з методом, у якого під капотом виконується SQL-запит або читання з файлу. Так досягається абстракція і прозорість.

інтерфейси

Завдання інтерфейсу - знизити рівень залежності сутностей друг від друга, додавши більше абстракції.

Не у всіх мовах присутній цей механізм, але в ООП мовами зі статичної типізацією без них було б зовсім зле. Вище ми розглядали абстрактні класи, зачіпаючи тему контрактів, зобов'язуючих імплементувати якісь абстрактні методи. Так ось інтерфейс дуже скидається на абстрактний клас, але є не класом, а просто пустушкою з перерахуванням абстрактних методів (без імплементації). Іншими словами, інтерфейс має декларативну природу, тобто, чистий контракт без крапельки коду.

Зазвичай в мовах, в яких є інтерфейси, немає множинного спадкоємства класів, але є множинне спадкування інтерфейсів. Це дозволяє класу перерахувати інтерфейси, які він зобов'язується імплементувати.

Класи з інтерфейсами складаються щодо «багато до багатьох»: один клас може імплементувати безліч інтерфейсів, і кожен інтерфейс, в свою чергу, може імплементуватиметься багатьма класами.

У інтерфейсу двостороннє застосування:

По один бік інтерфейсу - класи, імплементує даний інтерфейс.

По інший бік - споживачі, які використовують цей інтерфейс як опис типу даних, з яким вони (споживачі) працюють.

Наприклад, якщо якийсь об'єкт крім основного поведінки, може бути серіалізовані, то нехай він імплементує інтерфейс «Серілізуєме». А якщо об'єкт

можна зклонувати, то нехай він імплементує ще один інтерфейс - «клонують». І якщо у нас є якийсь транспортний модуль, який передає об'єкти по мережі, він буде приймати будь-які об'єкти, що імплементують інтерфейс «Серілізуєме».

ISP

(Interface Segregation Principle / Принцип поділу інтерфейсу / Четвертий принцип SOLID) закликає не створювати жирні універсальні інтерфейси. Замість цього інтерфейси потрібно розділяти на більш дрібні і спеціалізовані, це допоможе гнучкіше їх комбінувати в імплементують класах, які не змушуючи імплементувати зайві методи.

абстракція

В ООП все крутиться навколо абстракції. Існують фанатики, які стверджують, що абстракція повинна бути частиною ООП-трійці (інкапсуляція, поліморфізм, успадкування). А мій інспектор по УДО говорив протилежне: абстракція властива для будь-якого програмування, а не тільки для ООП, тому вона повинна стояти окремо. З іншого боку, те ж саме можна сказати і про інші принципи, але з пісні слів не викинеш. Так чи інакше, абстракція потрібна, і особливо в ООП.

рівень абстракції

Тут не можна не процитувати один відомий жарт:

- будь-яку архітектурну проблему можна вирішити додаванням додаткового шару абстракції, крім проблеми великої кількості абстракцій.

Неправильний вибір рівня абстракції веде до однієї з двох проблем:

якщо абстракції недостатньо, подальші розширення проекту будуть упиратися в архітектурні обмеження, які ведуть або до рефакторингу і зміні архітектури, або до великої кількості милиць (обидва варіанти зазвичай несуть за собою біль і фінансові втрати)

якщо рівень абстракції занадто високий, це призведе до оверінжінірінгу у вигляді надто складної архітектури, яку важко підтримувати, і зайвої гнучкості, яка ніколи в цьому проекті не стане в нагоді. У цій ситуації будь-які найпростіші зміни в проекті будуть супроводжуватися додатковою роботою для задоволення вимог архітектури (це теж часом несе певну біль і фінансові втрати)

Ще важливо розуміти, що рівень абстракції визначається не для всього проекту в цілому, а окремо для різних компонентів. У якихось місцях системи абстракції може бути недостатньо, а десь навпаки - перебір. Однак, невірний вибір рівня абстракції можна виправити своєчасним рефакторингом. Ключове слово - своєчасним. Запізнілий рефакторинг провести проблематично, коли на даному рівні абстракції реалізовано вже безліч механізмів. Проводити обряд рефакторінга в запущених системах може сполучатися з гострим болем у важкодоступних місцях програміста. Це приблизно як поміняти фундамент в будинку - дешевше побудувати поруч будинок з нуля.

Десятиліття розробки привели до того, що сформувався список найбільш часто вживаних архітектурних рішень, які згодом були класифіковані спільнотою, і стали називатися паттернами проектування. Саме тому, коли я прочитав вперше про патерни, я з подивом виявив, що виявляється, багато хто з них я вже використовую на практиці, просто не знав, що у цих рішень є назва.

Патерни проектування, як і абстракція, властиві не тільки ООП розробці, але і іншим парадигм

Призначення патернів - допомога у вирішенні архітектурних проблем, які або вже виявилися, або найімовірніше будуть виявлені в ході розвитку проекту. Так ось, прочитавши про патерни, у новачка може з'явиться нездоланний спокуса використовувати патерни не для вирішення проблем, а для їх породження. А оскільки розробник в своїх бажаннях розбещений, він може почати не вирішувати задачу за допомогою патернів, а підлаштовувати будь-які завдання під рішення за допомогою патернів.

Ще одна цінність від патернів - формалізації термінології. Набагато простіше колезі сказати, що в цьому місці використовується «ланцюжок обов'язків», ніж півгодини малювати поведінку і відносини об'єктів на папірці.

В умовах сучасних вимог наявність у вашому коді слова class не робить з вас ООП-програміста. Бо якщо ви не використовуєте механізми (поліморфізм, композицію, успадкування і т. Д.), А замість цього застосовуєте класи лише для

угруповання функцій і даних, то це не ООП. Те ж саме можна вирішити іншими можливостями синтаксису.

Будь-які описані механізми, принципи і патерни, як і ООП в цілому не варто застосовувати там, де це безглуздо або може нашкодити.

Опис таблиць бази даних

Для роботи з базою даних використовується Mongoose. Mongoose - бібліотека об'єктних даних моделювання (ODM) для MongoDB та Node.js. Він управляє зв'язками між даними, забезпечує перевірку схеми і використовується для перекладу між об'єктами в коді та представлення цих об'єктів у MongoDB.

Обмін об'єктів між Node та MongoDB, що управляється через Mongoose

MongoDB - бездокументальна база даних NoSQL. Це означає, що ви можете зберігати в ньому документи JSON, і структура цих документів може змінюватися, оскільки вона не застосовується, як SQL-бази даних. Це одна з переваг використання NoSQL, так як прискорює розробку додатків і зменшує складність розгортання на сервері.

Термінологія:

Колекції

"Колекції" в Mongo еквівалентні таблицям у реляційних базах даних. Вони можуть містити кілька документів JSON.

Документи

"Документи" еквівалентні записам або рядкам даних у SQL. Хоча рядок SQL може посилатися на дані в інших таблицях, документи Mongo зазвичай поєднують їх у документі.

Поля

"Поля" або атрибути схожі на стовпці таблиці SQL.

Схема

Хоча Mongo не має схеми, SQL визначає схему через визначення таблиці. "Схема" Мангуста - це структура даних документа (або форма документа), що застосовується через прикладний рівень.

Моделі

"Моделі" - це конструктори вищого порядку, які приймають схему і створюють екземпляр документа, еквівалентний записам у реляційній базі даних.

Middleware

Middleware - це функції, які виконуються на певних етапах конвеєра. Mongoose підтримує проміжне програмне забезпечення для таких операцій:

Сукупність

Документ

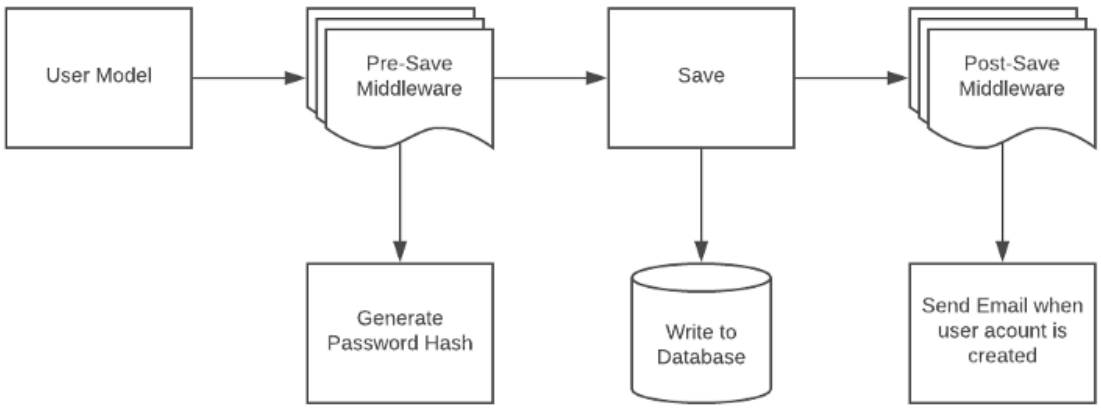
Модель

Запит

Наприклад, моделі мають функції до і після, які приймають два параметри:

Тип події ("init", " validate", " save", " remove")

Зворотний виклик, який виконується за допомогою цього посилання на екземпляр моделі



Структура таблиці “Користувач”

Ім’я	**Тип**	**Посилання**	**Опис**
_id	Schema.Types.ObjectId	-	Унікальний ідентифікатор для запису в базу даних
email	String(unique)	-	Адреса електронної пошти користувача

user_name	String(unique)	-	Логін користувача
password_hash	String	-	Хеш паролю

Структура таблиці “Регіон”

Ім’я	**Тип**	**Посилання**	**Опис**
_id	Schema.Types.ObjectId	-	Унікальний ідентифікатор для запису в базу даних
iso	String	-	Ідентифікаційной код регіону за системою ISO
region_name	String	-	Назва регіону
s_values	{ type: [Number] }	-	Масив значень за всі місяці, об’єму розсіяних променів сонця в регіоні
d_values	{ type: [Number] }		Масив значень за всі місяці, об’єму прямих променів сонця в регіоні

Структура таблиці “Маркер”

Ім’я	**Тип**	**Посилання**	**Опис**

_id	Schema.Types.ObjectId	-	Унікальний ідентифікатор для запису в базу даних
position.lat	Number	-	Широта координати, записана в об’єкт position
position.lng	Number	-	Довгота координати, записана в об’єкт position
system	Schema.Types.ObjectId	System	Посилання на об’єкт іншої таблиці, що дає можливість швидко отримати доступ.
system_type	Schema.Types.ObjectId	Stations/Panels	Посилання на об’єкт іншої таблиці, що дає можливість швидко отримати доступ.
square	Number	-	Площа системи
calculations	[{kWh:{type:Number}}]	-	Масив обрахунків, дані за кожним місяцем
region_iso	String	-	Ідентифікатор регіону в який

			попадає маркер

Структура таблиці “Кут”

Ім’я	**Тип**	**Посилання**	**Опис**
_id	Schema.Types.ObjectId	-	Унікальний ідентифікатор для запису в базу даних
lat	Number	-	Широта
angle	Number	-	Кут
ps_values	{ type: [Number] }	-	Масив значень за всі місяці на даній широті та куту, об’єму розсіяних променів сонця в регіоні

Структура таблиці “Панель[i]”

Ім’я	**Тип**	**Посилання**	**Опис**
_id	Schema.Types.ObjectId	-	Унікальний ідентифікатор для запису в базу даних
type	String	-	Тип панелі
name	String	-	Назва

efficiency	Number	-	ККД даного типу панелей

Структура таблиці “Станція[1]”

Ім’я	**Тип**	**Посилання**	**Опис**
_id	Schema.Types.ObjectId	-	Унікальний ідентифікатор для запису в базу даних
type	String	-	Тип панелі
name	String	-	Назва
efficiency	Number	-	ККД даного типу станцій
temperature	Number	-	Температура води

Розробка кабінету користувача

Кабінет користувача – це основне робоче місце користувача в системі. Він включає в себе інші підмодулі, які виконують основні функції системи, та є елементом компонування в системі.

Підмодулі кабінету користувача:

- Модуль авторизації
- Панель управління
- Карта
  - Модуль роботи з маркером

- Задання параметрів для маркеру
- Модуль обрахунків

- Модуль перегляду геліостанцій
- Модуль перегляду сонячних панелей
- Модуль перегляду регіонів
- Модуль перегляду кутів

Підомодуль автризації

- Форма реєстрації
- Форма авторизації
- Система доступу до даних в залежності від прав користувача

Підмодуль карта

- Перегляд карти
- Відображення фільтрів в залежності від характеристик регіону
- Перегляд маркерів
- Додавання маркеру
- Задання маркеру додаткових даних необхідних для обрахунків
- Можливість отримати розрахунки
- Можливість збереження маркету та обрахунків по ньому

Підмодуль відображення кутів/регіонів/маркерів було зроблено в вигляді таблиць, які зручно переглядати. Ми можемо швидко знати потрібний нам для перегляду елемент, та розгорнути для перегляду детальної інформації.

Висновки до розділу

В даному розділі було описано реалізацію системи, спосіб взаємодії між різними сервісами та компонентами, архітектуру серверу.

КОРИСТУВАННЯ СИСТЕМОЮ

При першому запуску програми, користувач повинен зареєструватись, чи пройти авторизацію: рис 5.1

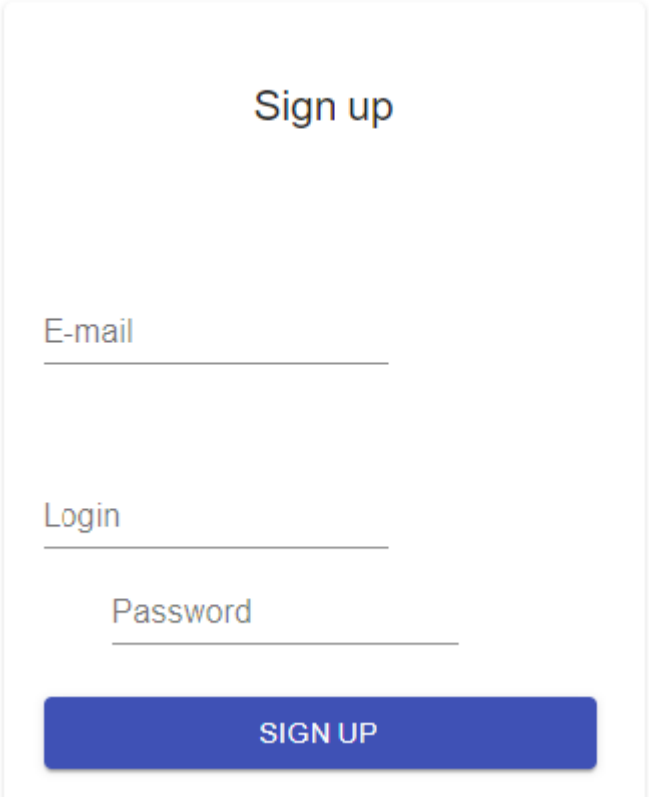


Рис 5.1 – реєстрація в системі

Після цього, він отримає можливість користуватись всіма можливостями веб-сервісу, а це: інтерактивна карта, вікна для перегляду даних з бд, типи систем та їх різновидності. Рис 5.2

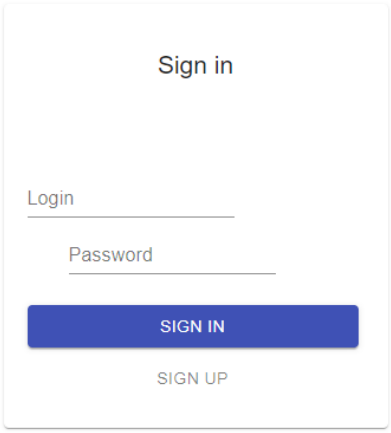


Рис 5.2 – Проходження авторизації

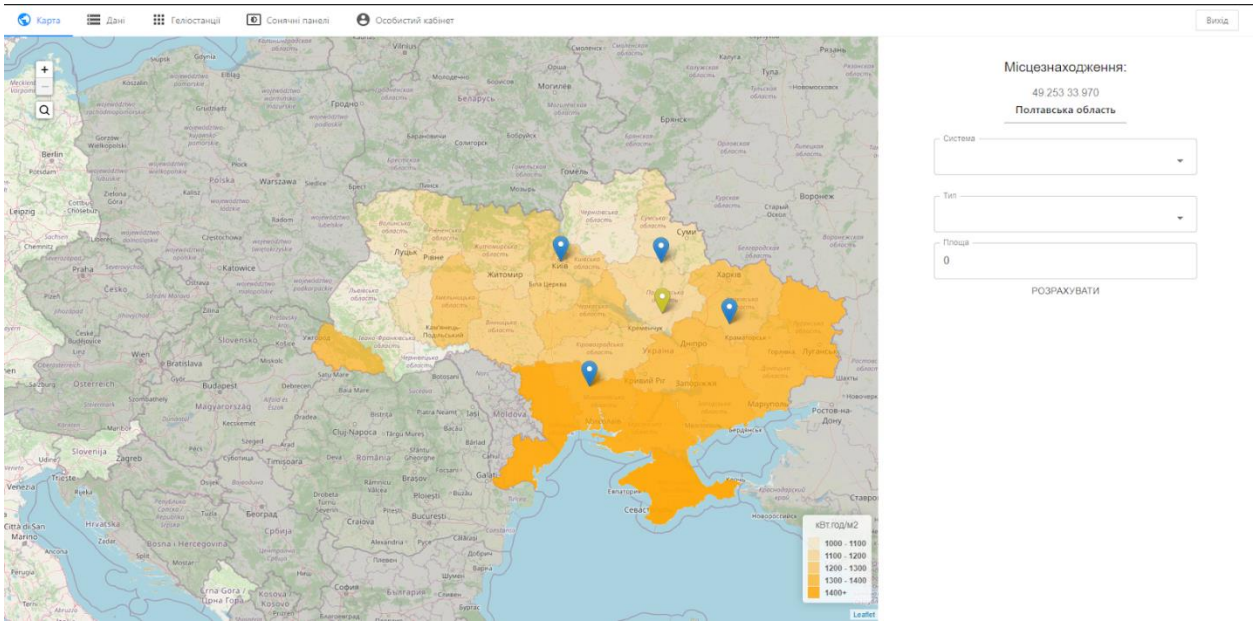


Рис 5.3 – вибираємо місцезнаходження, ставлячи маркер на карту

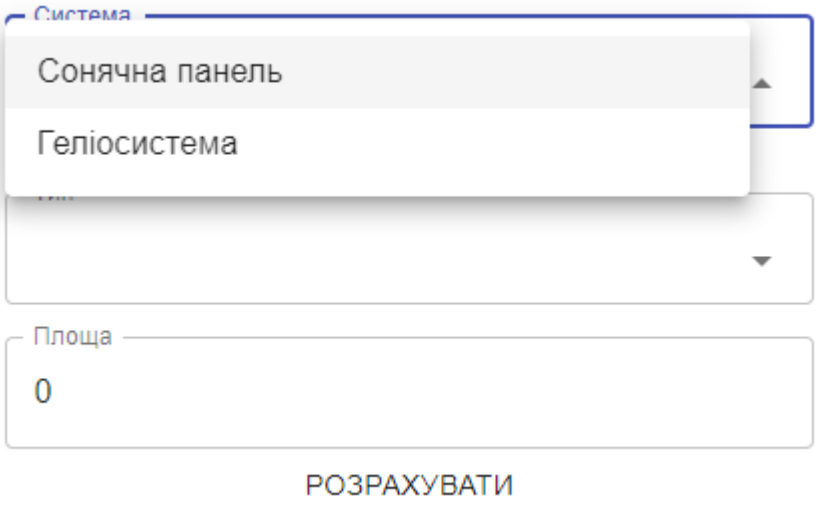


Рис 5.4 – Випадаючий список типів станцій

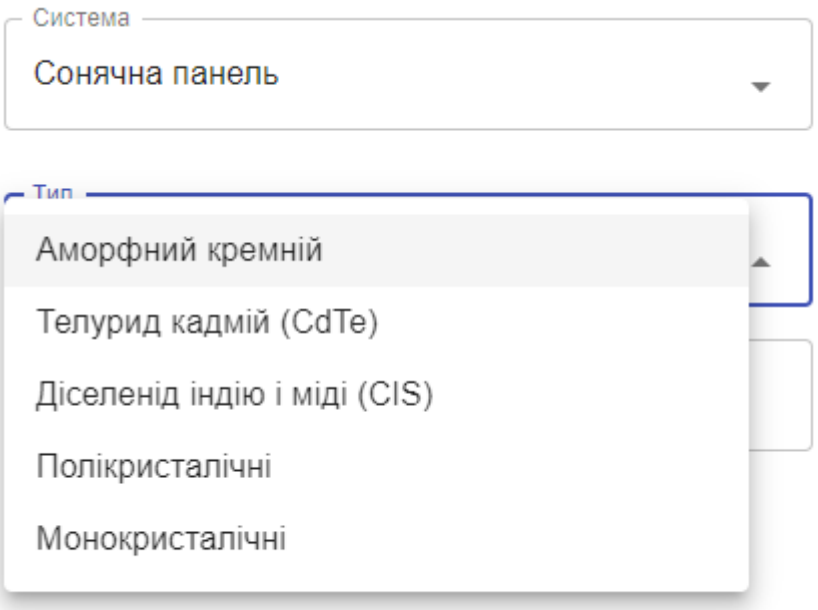


Рис 5.5 – вибираємо тип сонячної батареї

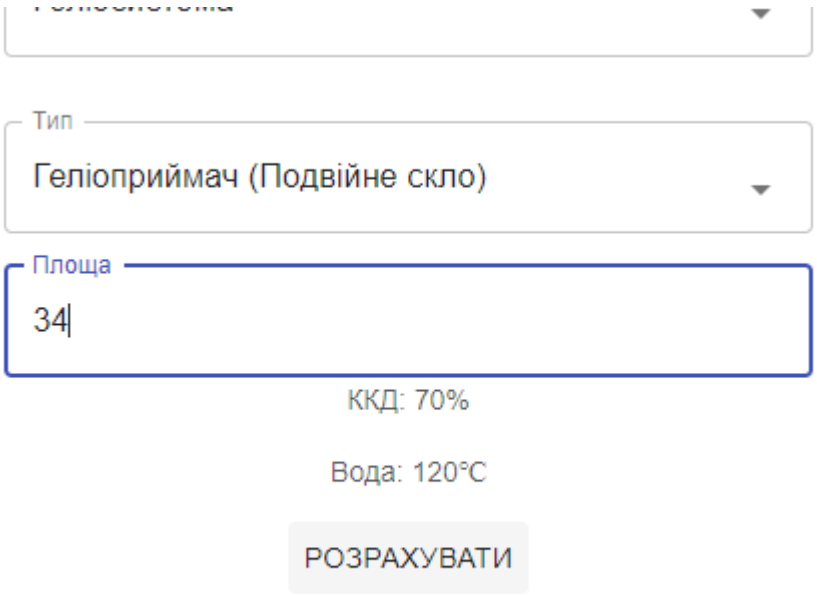


Рис 5.6 – форма розрахунку

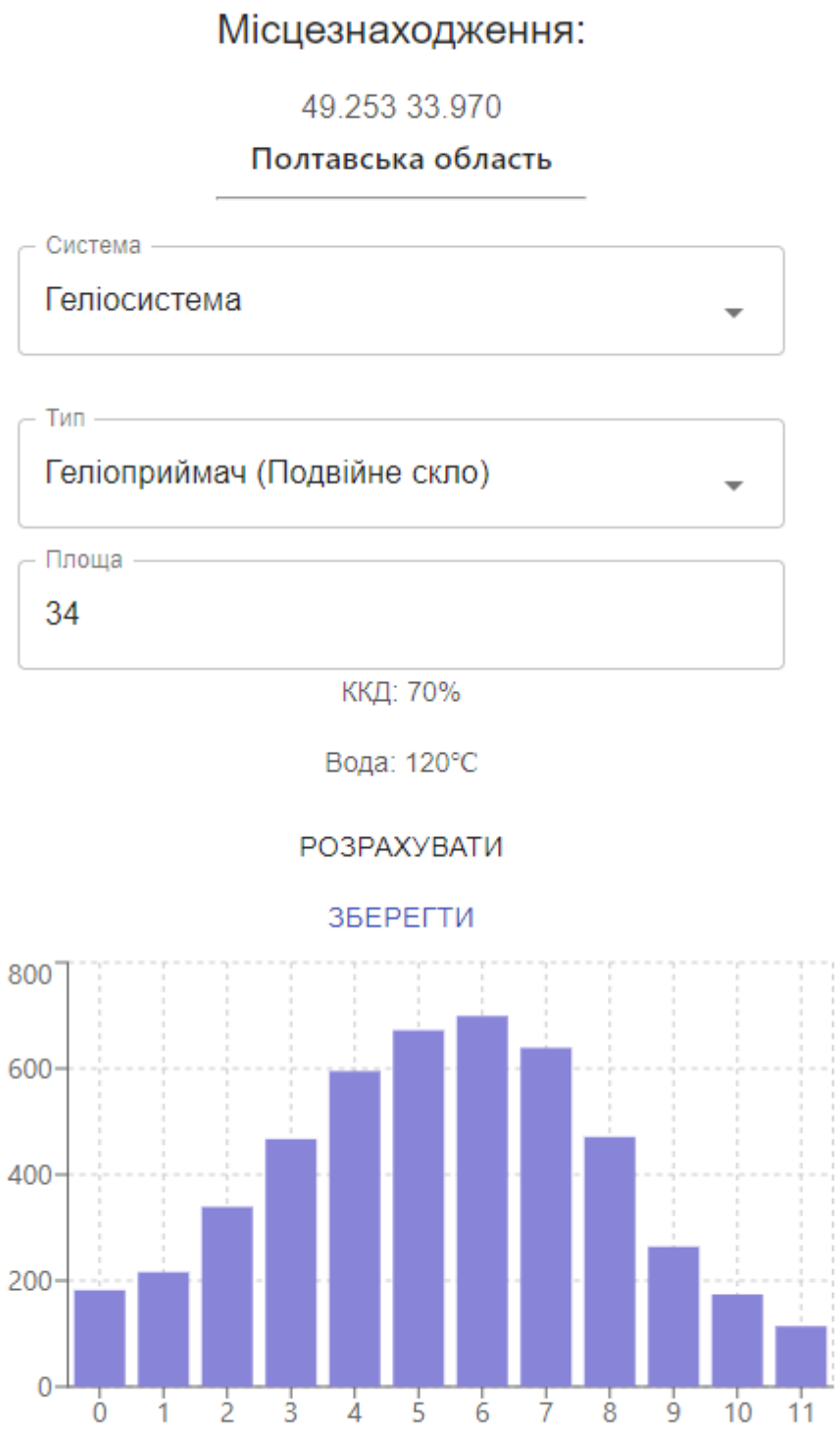


Рис 5.7 – Результат розрахунків у вигляді діаграми

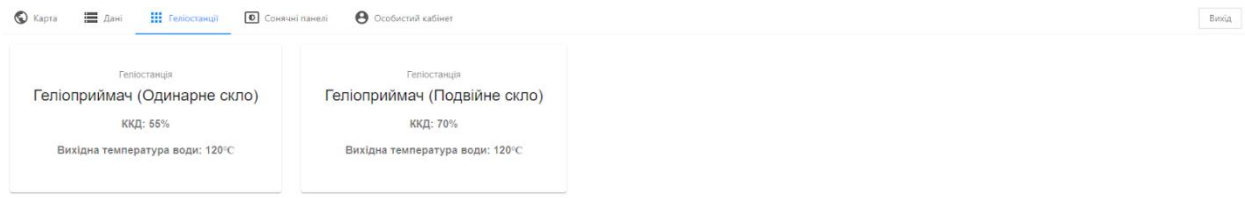
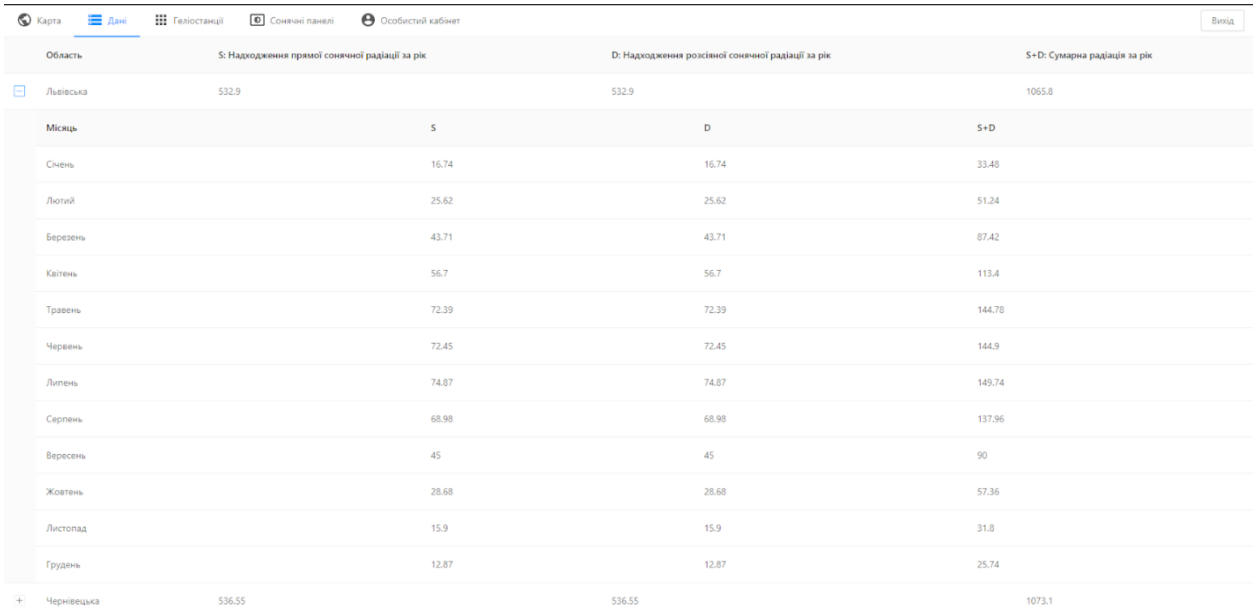


Рис 5.8 – різновиди геліостанцій заданих в системі



Область	S: Надходження прямої сонячної радіації за рік	D: Надходження розсіяної сонячної радіації за рік	S+D: Сумарна радіація за рік
Львівська	532.9	532.9	1065.8

Місяць	S	D	S+D
Січень	16.74	16.74	33.48
Лютий	25.62	25.62	51.24
Березень	43.71	43.71	87.42
Квітень	56.7	56.7	113.4
Травень	72.39	72.39	144.78
Червень	72.45	72.45	144.9
Липень	74.87	74.87	149.74
Серпень	68.98	68.98	137.96
Вересень	45	45	90
Жовтень	28.68	28.68	57.36
Листопад	15.9	15.9	31.8
Грудень	12.87	12.87	25.74

Чернівецька	536.55	536.55	1073.1

Рис 5.9 – Дані по областях

Висновки до розділу

В даному розділі було продемонстровано, як користувач повинен працювати з додатком.

ВИСНОВКИ

В ході аналізу існуючого програмного забезпечення для аналізу сонячної активності були досліджені системи, які служать для вирішення поставленої задачі. Аналіз показав, що існуючі системи не вирішують проблему в повному обсязі. Розроблений програмний продукт для користувачів, щоб взаємодіяти з картою в Інтернеті, аналізувати. Крім того, користувачі мають можливість використовувати онлайн-калькулятори, експериментувати з показниками в різних регіонах. Модуль написаний на JavaScript та за допомогою React та Leaflet.
Огляд методів та інструментів розробки програмних систем. Вибір створення програмної системи на основі веб-технологій, а також побудованої на трирівневій архітектурі виправданий. Це дозволяє підвищити гнучкість та зручність системи, як при розробці та обслуговуванні, так і у використанні.
За результатами тестових завдань було підтверджено правильність отриманих результатів, тому система відповідає вимогам. Результати, отримані користувачем, можуть мати помилку (10-20%), що залежить від специфіки області використання сонячних батарей та сонячних систем. Ця система значно спрощує процес моделювання системи виробництва енергії з альтернативних джерел і дозволяє користувачам отримати приблизний результат.
Користувачами системи може бути кожен, хто хоче використовувати сонячну енергію або хоче вивчити перспективи. Програмне забезпечення може використовуватися в будь-якій операційній системі, яка має браузер, який підтримує найновіші веб-стандарти, а також має постійний доступ до Інтернету.
Також ця система може використовуватися не тільки для сонячної енергії, але може бути розширена для використання інших видів альтернативних джерел енергії.

Перелік використаних джерел

1. С. О. Кудря. Нетрадиційні та відновлювані джерела енергії / Кудря С. О. − Підручник. – Київ: Національний технічний університет України («КПІ»), 2012.– 495с.

2. Н.М.Мхитарян. Энергетика нетрадиционных и возобновляемых источников. К., Наукова думка, 1999. – 314 с.

3. Даффи У.Дж., Бекман У.А. Тепловые процессы с использованием солнечной энергии /Под ред. Ю.Н.Малевского – М., 1977.

4. Твайделл Дж., Уэйр А. Возобновляемые источники энергии. – М.: Энергоатомиздат. 1990. – 344 с.

5. Научно-прикладной справочник по климату СССР, Сер. Вып. 10: УССР. Кн. 1. – Ч. 1. Солнечная радиация и солнечное сияние. – Л.: Гидрометеоиздат, 1990. – 608с.

6. Рекомендации по проектированию установок солнечного горячего водоснабжения для жилых и общественных зданий. Киев, КиевЗНИИЭП, 1987. – 119с.

7. Martin Fowler — GUI Architectures. Часть 1 [Електронний ресурс]. — 2009. — Режим доступу: http://www.rusdoc.ru/articles/18358/

8. Martin Fowler — GUI Architectures. Часть 2 [Електронний ресурс]. — 2009. — Режим доступу: https://habrahabr.ru/post/53536/.

9. Майер Л. — MAVLink Micro Air Vehicle Communication Protocol [Електронний ресурс]. — 2009. — Режим доступу: http://qgroundcontrol.org/mavlink/start.

ДОДАТОК А

Автоматизація бізнес процесів з використанням CRM системи Salesforce

Специфікація

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТМ62_20_19-1

Аркушів 1

Київ 2020

Позначення	Найменування	Примітки
Документація		
УКР.НТУУ”КПІ ”_ТЕФ_АПЕПС_ТМ62_20_9-1	Дипломна робота Дукач С.В.docx	Пояснювальна записка
Компоненти		
УКР.НТУУ”КПІ ”_ТЕФ_АПЕПС_ТМ62_20_9-2	StudiesTools.cmp MainDepartmentPage.cmp GeneralSchedule.cmp Relationships.cmp StudentInfo.cmp ProfesorInfo.cmp Authorization.cmp FAQ.cmp SelectDepartment.cmp SelectDepartmentController.apxc ProfessorInfoController.apxc StudentInfoController.apx ScheduleController.apx	Основні компоненти
УКР.НТУУ”КПІ ”_ТЕФ_АПЕПС_ТМ62_20_9-3	Додаток В	Опис програмного модуля

ДОДАТОК Б

Інтерактивна карта розподілу потенціалу сонячного гарячого водопостачання по областях України

Код програми

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТМ62_20_9-2

Аркушів 10

Київ 2020

```
var crypto = require("crypto");

const UserModel = require("../models/User");

const createJWToken = require("../utils/createJWToken");
const verifyJWToken = require("../utils/verifyJWToken");

class UserController {
  show = (req, res) => {
    const id = req.params.id;
    UserModel.findById(id, (err, user) => {
      if (err) {
        return res.status(404).json({
          message: "User not fount",
        });
      }
      res.json(user);
    });
  };

  getMe = (req, res) => {
    const token = req.headers.token;
    verifyJWToken(token)
      .then((user) => {
        console.log("user", user.data._doc);
        res.json(user);
      })
      .catch((err) => {
```

```
      res.status(403).json({ message: "Invalid auth token provided." });
    });
};

signup = (req, res) => {
  const postData = {
    email: req.body.email,
    username: req.body.username,
    password: req.body.password,
  };

  this.create(postData, res);
};

create = (postData, res) => {
  const user = new UserModel({
    email: postData.email,
    username: postData.username,
    password_hash: crypto
      .createHash("md5")
      .update(postData.password)
      .digest("hex"),
  });
  console.log("object");
  user
    .save()
    .then((obj) => {
      if (res) {
```

```
        res.json({
          status: "success",
          obj,
        });
      }
    })
    .catch((reason) => {
      if (res) {
        res.status(500).json({
          status: "error",
          message: reason,
        });
      }
    });
};

login = (req, res) => {
  const postData = {
    username: req.body.username,
    password: req.body.password,
  };

  console.log(postData);

  UserModel.findOne({ username: postData.username }, (err, user) => {
    console.log(
      user.password_hash,
      crypto.createHash("md5").update(postData.password).digest("hex")
```

```
    );
    if (
      user.password_hash ===
      crypto.createHash("md5").update(postData.password).digest("hex")
    ) {
      const token = createJWToken(user);

      res.json({
        status: "success",
        token,
      });
    } else {
      res.status(403).json({
        status: "error",
        message: "Incorrect password or username",
      });
    }
  });
 };
}

exports.UserController = UserController;
var crypto = require("crypto");

const RegionModel = require("../models/Region");

const regionsJson = require("../assets/data/regions.json");
```

```
// const createJWToken = require("../utils/createJWToken");
// const verifyJWToken = require("../utils/verifyJWToken");

class RegionController {
  constructor() {
    RegionModel.find({}, (err, regions) => {
      if (err || !regions || regions.length === 0) {
        regionsJson.map((item) => {
          const region = new RegionModel(item);

          region
            .save()
            .then((obj) => {
              console.log("fads", obj);
            })
            .catch((reason) => {});
        });
      }
    });
  }

  getAll = (req, res) => {
    RegionModel.find({}, (err, regions) => {
      console.log(regions);
      if (err || !regions) {
        res.status(500).json({ status: "error", message: err });
      }
      res.json(regions);
```

```
    });
  };
}

exports.RegionController = RegionController;
var crypto = require("crypto");

const MarkerModel = require("../models/Marker");

// const createJWToken = require("../utils/createJWToken");
// const verifyJWToken = require("../utils/verifyJWToken");

class MarkerController {
  create = (postData, res) => {
    const marker = new MarkerModel(postData.body.marker);
    console.log(postData.body.marker);
    marker
      .save()
      .then((obj) => {
        console.log("fads", obj);
        if (res) {
          res.json({
            status: "success",
            obj,
          });
        }
      })
      .catch((reason) => {
```

```
      if (res) {
        res.status(500).json({
          status: "error",
          message: reason,
        });
      }
    });
  };

  getAll = (req, res) => {
    MarkerModel.find()
      .populate("creator")
      .exec((err, markers) => {
        console.log(markers);
        if (err || !markers) {
          res.status(500).json({ status: "error", message: err });
        }
        res.json(markers);
      });
  };
}

exports.MarkerController = MarkerController;
const bodyParser = require("body-parser");
const cors = require("cors");

const UserCtrl = require("../controllers/UserController").UserController;
const MarkerCtrl = require("../controllers/MarkerController").MarkerController;
```

```
const RegionCtrl = require("../controllers/RegionController").RegionController;
// const ProfileCrtl = require("../controllers/ProfileController")
//   .ProfileController;
// const RatingCtrl = require("../controllers/RatingController").RatingController;

const checkAuth = require("../middlewares/checkAuth");

const createRoutes = (app) => {
  const UserController = new UserCtrl();
  const MarkerController = new MarkerCtrl();

  const RegionController = new RegionCtrl();
  // const ProfileController = new ProfileCrtl();
  // const RatingController = new RatingCtrl();

  app.use(function (req, res, next) {
    res.header("Access-Control-Allow-Origin", "*");
    res.header("Access-Control-Allow-Credentials", true);
    res.header("Access-Control-Allow-Methods", "GET,PUT,POST,DELETE,OPTIONS");
    res.header(
      "Access-Control-Allow-Headers",
      "Origin,X-Requested-With,Content-Type,Accept,content-type,application/json"
    );
    next();
  });

  app.use(cors());
  app.use(bodyParser.json());
```

```
  app.use(checkAuth);

  app.post("/user/signup", UserController.signup);
  app.post("/user/signin", UserController.login);
  app.get("/user/me", UserController.getMe);
  // app.get("/user/:id", UserController.show);

  app.post("/marker/create", MarkerController.create);
  app.get("/marker/getall", MarkerController.getAll);

  app.get("/region/getall", RegionController.getAll);
  // app.post("/match/get", MatchController.get);
  // app.post("/match/addgame", MatchController.addGame);
  // app.post("/match/updategame", MatchController.updateGame);
  // app.post("/match/getgames", MatchController.getGames);
  // app.post("/match/getmatches", MatchController.getMatches);

  // app.post("/profile/info", ProfileController.getInfo);
  // app.post("/profile/matches", ProfileController.getMatches);

  // app.post("/rating/get", RatingController.getRating);
};

module.exports = createRoutes;
```

ЗМІСТ

ЗАГАЛЬНІ ВІДОМОСТІ

У додатку розглядається один з програмних модулів системи — модуль для роботи з веб-сторінкою з кодом УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТМ62_20_19-1, що міститься у файлі SunMap. Модуль реалізовано за допомогою бібліотек React та Js. Модуль призначений для управління системою, яка відповідають за керування метеоданими, які надходять з API. Користувач має можливість обирати вже існуючі записи або створити новий та розрахувати дані.

ФУНКЦІОНАЛЬНЕ ПРИЗНАЧЕННЯ

Призначенням модулю для роботи з веб-картоє є відображення інформації х серверу та бази даних та безпосереднього прийняття даних введення та обмін інформацією із клієнтським інтерфейсом. Використання такого шаблону дозволяє створювати програмне забезпечення, де інтерфейс і логіка роботи модуля для бази даних та серверу є незалежними компонентами, що дає можливість використовувати його для зменшення навантаження на клієнтську частину системи.

ОПИС ЛОГІЧНОЇ СТРУКТУРИ

Слідкувати за зміною інформації про підсистеми є головним завданням модуля. При запуску системи модуль повертає всю інформацію, яка міститься в базі даних, для відображення даних на користувацькому інтерфейсі. Також модуль оброблює інформацію, надаючи змогу вводити потрібні дані та проводити обчислення та відображати результати у вигляді таблиці.

ТЕХНІЧНІ ЗАСОБИ ЩО ВИКОРИСТОВУЮТЬСЯ

Модуль розроблено у середовищі розробки Microsoft Visual Studio Code, що забезпечує набір сервісних функцій та графічний діалог з користувачем, на комп’ютері, має встановлений браузер та підключення до інтернету. Для реалізації клієнтської частини було використано бібліотеки React та Leaflet. За допомогою елементів React клієнтська частина може з’єднуватись з сервером за допомогою API.

В якості провайдера самої інтерактивної карти було обрано відкритий сервіс OpenStreetMap.

ВИКЛИК І ЗАВАНТАЖЕННЯ

Програмний модуль реалізований як окремий клас, який забезпечує існування клієнтської частини та бізнес-логіки окремо від одного, але разом із цим запуск обох компонентів відбувається одночасно.

Для використання цього модуля треба завантажити веб-сторінку, яка зробить потрібні запити та відобразить результат.

ВХІДНІ І ВИХІДНІ ДАНІ

Вхідними даними для модуля є інформація, яку користувач вводить в додатку та місцеположення.

Вихідними даними програмного модуля є результати обчислення, які відображаються у вигляді таблиці.

ДОДАТОК Г

Інтерактивна веб-карта для представлення енергоресурсів та об’єктів відновлюваної енергетики України (розробка веб-інтерфейсу користувача).

Довідки про впровадження результатів роботи

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТІ51172_19Б 14-1

Аркушів 1

Київ 2019